2019 IEEE Intelligent Transportation Systems Conference (ITSC)
Auckland, NZ, October 27-30, 2019

Vision-Based Trajectory Planning via Imitation Learning for Autonomous Vehicles

Peide Cai, Yuxiang Sun, Yuying Chen, Ming Liu

***Abstract*— Reliable trajectory planning like human drivers in real-world dynamic urban environments is a critical capability for autonomous driving. To this end, we develop a vision and imitation learning-based planner to generate collision-free trajectories several seconds into the future. Our network consists of three sub-networks to conduct three basic driving tasks: *keep straight*, *turn left* and *turn right*. During the planning process, high-level commands are received as prior information to select a specific sub-network. We create our dataset from the Robotcar dataset, and the experimental results suggest that our planner is able to reliably generate trajectories in various driving tasks, such as turning at different intersections, lane-keeping on curved roads and changing lanes for collision avoidance.**

I. INTRODUCTION

In order to reach the level of autonomous driving, a vehicle needs to model the surroundings using perception techniques, such as 3-D SLAM [1]–[3]. Then the extracted information could be used to plan collision-free trajectories to the goal position [4]. However, it is still a challenge to enable autonomous vehicles to catch up with human drivers' capabilities, especially in real-world environments, where trajectory planning is a crucial task.

Within the trajectory planning framework, the solution trajectory is represented as a time-parametrized function $\pi(t) : [0,T] \rightarrow \mathcal{X}$, where T is the planning horizon and $\mathcal{X}$ is the configuration space of the vehicle [5]. For human drivers who plan to reach a destination in an unknown environment, they are usually informed by a navigation software about which direction to drive, such as keeping straight or turning left at the next intersection. Based on this phenomenon, we propose a vision-based imitation learning method to plan trajectories with high-level driving commands (*keep straight*, *turn left*, *turn right*).

Basically, methods for planning and decision-making for autonomous vehicles can be divided into three approaches: traditional sequential planning, end-to-end control and end-to-end planning. Their pipelines are illustrated in Fig. 1.

The traditional approaches have been regarded as an optimization problem in which the optimal actions given a cost function should be selected. Various approaches of them differ in the shape of both actions and cost functions [6]. However, in most work they have to be carefully designed. Besides, many of them are based on map-building, which need to be updated dynamically with difficulty under the limitation of computational resource, which may not be quick enough to reflect changes in environment [7], [8]. Additionally, the traditional approaches classically rely on a perception system to extract information in the form of manually designed features from raw sensory input. However, they are constrained by the adaptivity to generic environments [8].

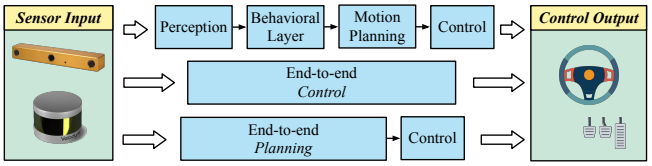


Fig. 1. Different approaches for trajectory planning and decision-making on autonomous vehicles.

Recently, a number of methods based on deep learning integrating perception, planning and control have been proposed and achieved impressive results on vehicle navigation, called *end-to-end control* [9]–[11]. However, these prior efforts mostly express the problem as learning a mapping from perceptual inputs directly to vehicle control commands (steering angle and throttle). The deficiencies of this treatment are twofold. (i) The vehicle is not steerable at an intersection. Since the vehicle considers only the camera input for decision making, it may take a wrong turn for the lack of high-level navigation commands [12]. (ii) The learned behavioral policy can only be well performed on data collected with the specifically calibrated actuation setup [13].

In this paper, to tackle the problems above, we propose a novel network to map visual perception and state information into future trajectories with the help of high-level turning command. Here the modules of perception, behavioral layer and motion planning in traditional sequential planning framework are integrated together to form an *end-to-end planning* method. The proposed network has been successfully trained on our dataset created from the Robotcar [14], and proved to be reliable by comparing with different baselines. The main contributions of this paper are summarized as follows:

- By imitating the behavior of human drivers, we introduce a novel mapless learning-based planning method for autonomous vehicles in different occasions, with less prior information than traditional approaches.
- Our method is reliable when turning at an intersection and some other tough scenes such as changing lanes

This work was supported by the National Natural Science Foundation of China (Grant No. U1713211); the Research Grant Council of Hong Kong SAR Government, China, under Project No. 11210017, and No. 21202816, awarded to Prof. Ming Liu.

Peide Cai, Yuxiang Sun, Yuying Chen and Ming Liu are with the Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Hong Kong, China. (e-mail: pcaiaa@connect.ust.hk; sun.yuxiang@outlook.com; ychenco@connect.ust.hk; eelium@ust.hk)

for collision avoidance, slowing down for a slow car and lane-keeping on curved roads.
- The planned trajectories from our network can be further translated into steering and throttle commands by controllers designed for different vehicles, which is more generalized than the end-to-end control methods.

II. RELATED WORK

A. End-to-end Control Methods

ALVINN [15] in 1989 was the pioneer attempt to use a shallow neural network for autonomous driving. Since the method was very simple and comprised of limited number of neural network layers, it can only drive in very simple scenarios with few obstacles, far away from being applied in a real traffic environment. Even though, it demonstrated the potential for a well-trained end-to-end neural network navigating a vehicle on public roads.

Inspired by ALVINN, another framework called DAVE-2 was proposed by NVIDIA [9] in 2016 using a similar idea but benefitting from more powerful modern Convolution Neural Networks(CNN), more available data sets for training and higher computing power from GPUs. During driving, the camera mounted in the center portion behind the windshield was used to get the images in front of the car and transmit them into CNN to compute steering commands. By using CNN to learn lane following task on the road without manual intervention, this framework achieved extraordinary success in relatively simple real-world scenarios, such as flat or barrier-free roads. However, this framework was only preliminary, and more work needs to be done to improve system robustness.

Actually, those work mainly solves the problem of lane-following tasks. In order to resolve the ambiguity occurred at intersections and extend the problem into a goal-directed navigation task, [12] and [16] redesigned the network to receive not only perceptual inputs but also the driver's intention. The neural-network motion controller in [16] trained on data from the first floor of a building can also drive the robot successfully on the second floor, which has a different geometric arrangement and visual appearance.

B. End-to-end Planning Methods

To make the driving model more generic, recently some new methods have been approached as an end-to-end planning method leaving the control component outside the neural network [17], [18]. To investigate whether neural networks are capable of predicting the path 5 seconds into the future, [17] tested different networks with various inputs, such as a sequence of gray-scale images, past ego motion, detections of surrounding objects and lane marking estimations. All models were trained via imitation learning using the collected 7-hours real-world driving data in rural Europe. The results showed that the path predicted by LSTM or CNN-LSTM is smooth and feasible in many situations. However, the network was not supposed to handle decision-making and only lane-keeping task was considered. In addition, the interaction to the surroundings was not taken into account because of the data limitation. The ChauffeurNet proposed in [18] is a partially end-to-end planning method because the perception module is independent of the planning network, which processes raw sensor inputs into top-down representations of the environment. These mid-level inputs are received by a recurrent neural network (RNN) to compute a driving trajectory, after which a control module is designed to translate the trajectory into steering and acceleration. By augmenting their data with synthesized perturbations and augmenting the imitation loss with losses that discourage bad behavior, this method received good results on driving tasks such as stopping for stop-signs and lane-following along straight or curved roads. Different with [18], our planning architecture is mapless and fully end-to-end.

III. METHODOLOGY

A. Network Architecture

The goal of our work is to plan trajectories like those planned by humans. So we assume that the history driving trajectories before the current time are planned by humans and we train our network by learning from those trajectories. At every time step, the inputs of the network are camera images $\{\boldsymbol{i}^1,...,\boldsymbol{i}^K\}$ and vehicle's motion state information $\{\boldsymbol{m}^1,...,\boldsymbol{m}^K\}$ in the past 1.5 seconds (assume K steps). The output of the network is a collision-free trajectory by looking 3 seconds (assume P steps) into the future $\{\boldsymbol{m}^1_{plan},...,\boldsymbol{m}^P_{plan}\}$. We have three sub-networks to be chosen by the prior command information $\boldsymbol{c}$ to conduct different driving tasks. The high-level command information contains *"Turn left at the coming intersection"*, *"Keep straight"* and *"Turn right at the coming intersection"*. The movement information contains the lateral $\boldsymbol{x}$ and longitudinal $\boldsymbol{z}$ positions together with speed $\boldsymbol{v}$, which can be easily acquired and corrected by GPS, IMU, etc. Note that the positions are transformed from the world frame to the local one. Once a specific sub-network is chosen, the images will first be processed respectively by an image module $I(\boldsymbol{i})$, which is implemented as a convolutional network. It extracts a feature vector of length 128 from each image. Then a balance module $B(\boldsymbol{m})$ expands the length of each history movement vector from 3 to 32 to balance the influence of the vision and the movement feature vectors, where we draw on the experience of [16]. The balance module is implemented as a fully connected network. The outputs of these two modules are then concatenated together at every time step into joint vectors of length 160, represented by:

$$\boldsymbol{J^n} = < I(\boldsymbol{i}^n), B(\boldsymbol{m}^n) >,\ 1 \le n \le K \tag{1}$$

where $I(\boldsymbol{i}^n)$ is the output of the image module for the n-th image, $B(\boldsymbol{m}^n)$ is the output of the balance module for the n-th movement vector. And $\boldsymbol{J^n}$ is the n-th joint vector. Then the set of $\boldsymbol{J} = \{\boldsymbol{J}^1,...,\boldsymbol{J}^K\}$ will be processed by a LSTM network and further reshaped by a fully connected layer to get the final output vector of length $(3 \times P)$, of which every 3 cells represent a state $\boldsymbol{m}_{plan}$ at a time step in the preview horizon. Here $\boldsymbol{m}_{plan}$ also consists of $\boldsymbol{x}, \boldsymbol{z}$ and $\boldsymbol{v}$.

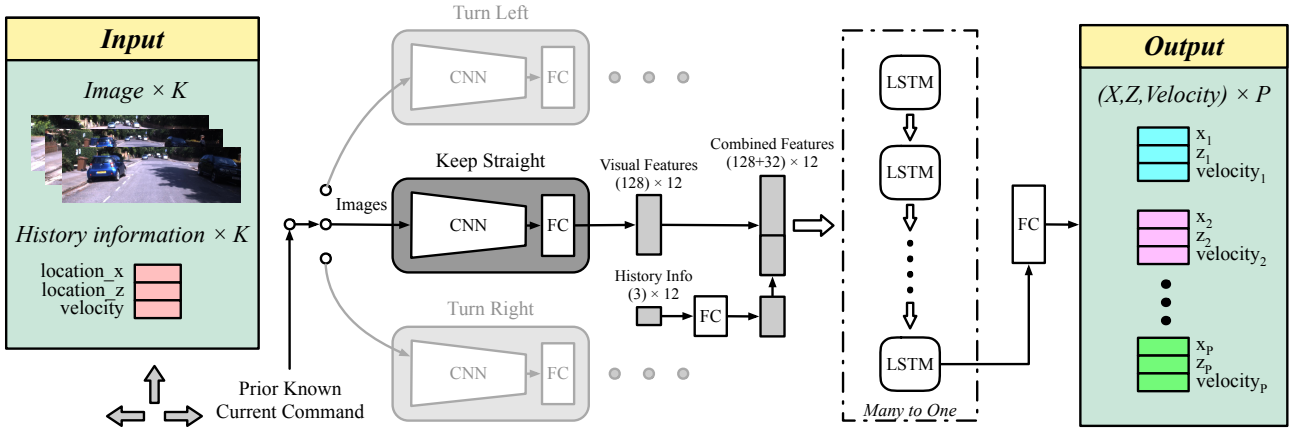


Fig. 2. The CNN-LSTM+State network architecture for trajectory planning. The command information takes three discrete values: Turn Left, Turn Right and Keep Straight. Then different sub-networks will be chosen to compute a trajectory. Note that the three network architectures based on different commands are identical, and they only differ in the training data. The figure is best viewed in color.

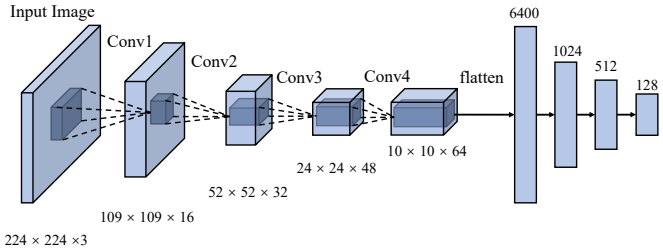


Fig. 3. The CNN module for processing every image at 244×244 pixel resolution. The output feature is a single vector of length 128 after four convolutional layers and three fully connected layers. Between every two convolutional layers there is a 2×2 max pooling layer.

Fig. 2 shows our proposed CNN-LSTM+State network architecture. Overall, the deep network planner F essentially performs a multivariable regression task, which can be formulated by the following equation:

$$\{\boldsymbol{m}^1_{plan},...,\boldsymbol{m}^P_{plan}\} = F\left(\{\boldsymbol{i}^1,...,\boldsymbol{i}^K\},\{\boldsymbol{m}^1,...,\boldsymbol{m}^K\}|\boldsymbol{c}\right) \quad (2)$$

B. Network Details

The CNN module for processing the input RGB images is shown in Fig. 3, which consists of four convolutional layers, four max pooling layers and three fully connected layers. For the four convolutional layers, the kernel sizes are 7, 6, 5, 5 and the corresponding numbers of filters are 16, 32, 48, 64. The strides are 1 for the four layers. The features are then flattened to get a single vector of length 6400. Then a multilayer perceptron of three fully connected layers transforms it into the final output feature vector of length 128. We performed batch normalization [19] after all hidden layers. In the CNN module, we use ReLU activation functions after each convolutional layer.

For the LSTM module, the number of features in the hidden state is 512. The number of recurrent layers is set to 3, which means that three LSTMs are stacked together, of which each LSTM receives the output of the upper one with the third LSTM layer giving the results.

For each training sample, given the ground truth trj_{gt} and the planned $(3\times P)$ vector trj_{plan}, our loss function is defined as follows:

$$\ell\left(trj_{plan},trj_{gt}\right) = \sum_{n=1}^{P}\{(\boldsymbol{x}^n_{plan}-\boldsymbol{x}^n_{gt})^2 + (\boldsymbol{z}^n_{plan}-\boldsymbol{z}^n_{gt})^2 + (\boldsymbol{v}^n_{plan}-\boldsymbol{v}^n_{gt})^2\} \quad (3)$$

C. Data Collection and preprocessing

1) Data Collection: We create our dataset from the Robotcar dataset, which is recorded in urban areas through 6 cameras mounted to the vehicle, along with LiDAR, GPS and INS information. The driving information in this dataset is over 1000 km containing many turns at crossroads and interactions with dynamic objects on the road, such as slowing down for a slow or parked car and changing lanes for collision avoidance. The ground truth position information can be acquired from the fused GPS+Inertial solution recorded at a frequency of 50Hz, which is also equipped in the dataset.

From Robotcar, we extract camera images and position information to train and test the planning network. To visualize and measure the performance, we need to project the trajectories and neighboring objects into a top-down view at every frame. In order to achieve this goal, it is necessary to detect and compute the position, orientation and size of each dynamic objects around the ego-vehicle. Based on the above two purposes, the stereo camera on top of the vehicle in Robotcar is the source where we pick pairwise images, of which the left and right cameras with a baseline of 24cm are used. We can then compute disparity and restore the depth of the environment with those paired images.

The data distribution and splitting are shown in Fig. 4. The final training dataset contains 52,200 images (left camera only) for about 2 hours of driving in daytime. We labelled every image with different commands telling the vehicle where to go based on the ground truth trajectory. Before we get the final dataset, we removed the sequences with poor GPS signals to ensure the quality of the training data. Note

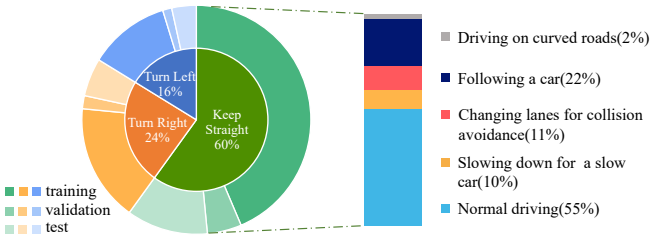


Fig. 4. Data distribution and splitting. Totaling 52,200 images for three driving tasks. For each situation, The split ratio for training, validation and test is 35:4:11. The staked column bar on the right shows the distribution of data in *keeping straight* situation, where we have balanced the portion for normal and abnormal driving cases. The turning situation does not contain much interaction with other road agents. The figure is best viewed in color.

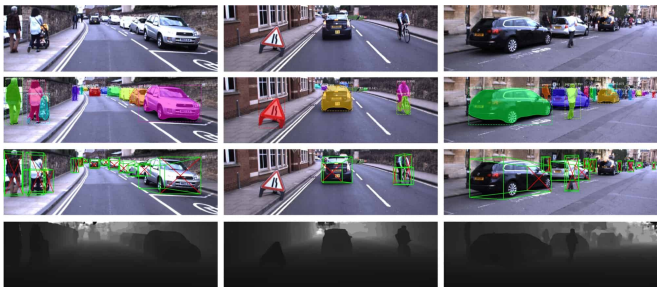


Fig. 5. The pipeline to extract object information around the ego-vehicle. Images in row 1 come from the left camera. Row 2 shows the detected objects from Mask R-CNN. Row 3 shows the estimated orientation and size from PSMNet. Row 4 shows the restored depth information for left camera images. The figure is best viewed in color.

that in this dataset, we do not consider the situation where the car stopped for traffic lights or stop signs.

2) Data Preprocessing: To train the network, we need to equip every image with the trajectory of ego-vehicle in the past 1.5 seconds and the next 3 seconds. We first interpolate the ground truth UTM (Universal Transverse Mercator) position and velocity series to the image timestamps recorded at 15Hz. For every image, the corresponding UTM coordinates ranging 4.5 seconds are then converted to the local coordinate system.

To visualize the road objects in analyzing the planning performance, we design a pipeline (Fig. 5) to extract the surrounding objects' size, orientation and position relative to the ego-vehicle. We first use the Mask R-CNN [20], [21] to detect different objects in every image frame and then the acquired 2D bounding boxes are used to estimate the objects' size and orientation [22], [23]. Then the pyramid stereo matching network (PSMNet) [24] is utilized to compute the depth information for left camera images. Finally, the objects' positions are computed by the depth and detected mask along with the camera parameters.

IV. Experiments

We train the three sub-tasks for *keep straight*, *turn left* and *turn right* separately in our **CNN-LSTM+State** network, which is implemented in Pytorch. We use the Adam optimizer [25] for training with the initial learning rate of

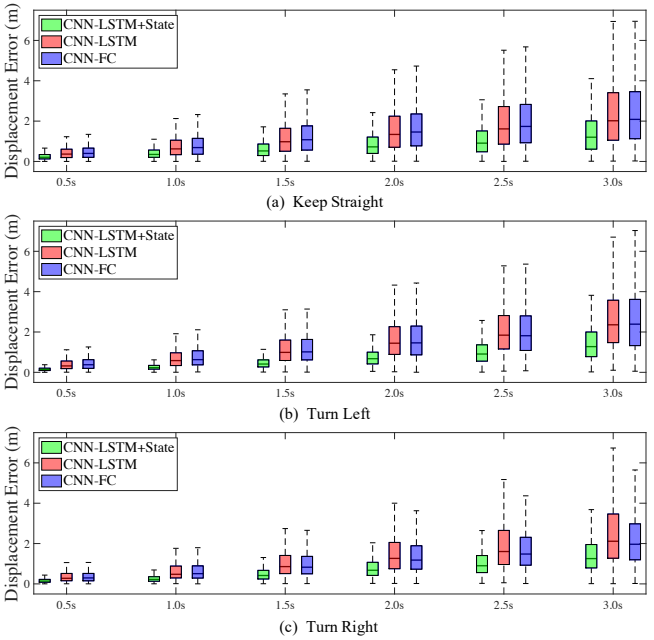


Fig. 6. Box plots on different scenes containing information on distribution of displacement error over the preview time in 3 seconds. The CNN-FC, CNN-LSTM and our full CNN-LSTM+State method are used to compute the results. The x-axis shows the preview time in seconds. The figure is best viewed in color.

0.001 and batch size of 32. We also use the same dataset and training procedure for other variants of our method. Compared with our full method CNN-LSTM+State, these models do not use the previous state information for planning trajectories:

- **CNN-FC**: Use image series as input by looking back 1.5 seconds, with CNN as visual encoder and fully connected layers (FC) as decoder.
- **CNN-LSTM**: Use image series as input by looking back 1.5 seconds, with CNN as visual encoder and LSTM as decoder.

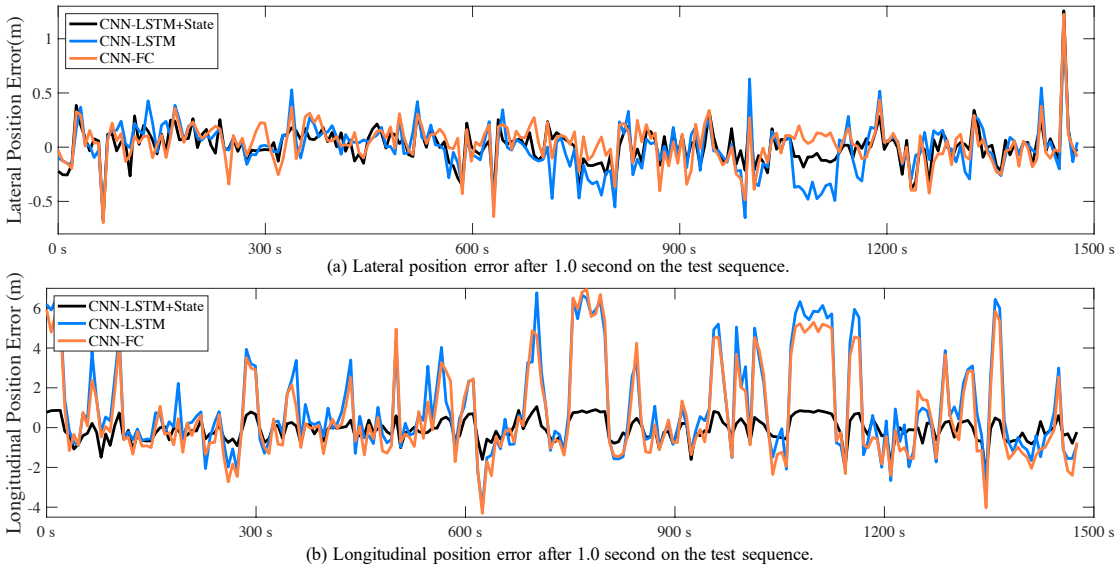
A. Quantitative Evaluation

1) Metrics: We use several metrics to measure the performance of different methods. All of the metrics are the average value over all test samples. $E(\ell_2)$ is the average displacement error [26], which is the mean distance over all planned points and the true points of a trajectory. Similarly, $E(lateral)$ and $E(longi)$ represent the lateral and longitudinal error. Besides, $E(final\ disp.)$ is the final displacement error [26], meaning the distance between the planned and true final destination of a trajectory. $E(speed)$ measures the mean speed error in the preview horizon. The average time for different networks to plan a trajectory is also recorded. We expanded the future trajectory for every frame into a driving area (D) according to the vehicle's width and then calculated the IoU based on the following equation:

$$IoU = \frac{D^{gt} \bigcap D^{plan}}{D^{gt} \bigcup D^{plan}} \tag{4}$$

TABLE I
QUANTITATIVE RESULTS BY DIFFERENT METHODS. SMALLER VALUES ENCODE BETTER PERFORMANCE EXCEPT FOR IOU.
(THE UNIT FOR T IS ms, FOR IoU IS %, FOR $E(speed)$ IS m/s, FOR THE REMAINING ERROR METRICS ARE m.)

Scenes	Methods	T	DLJ	IoU	$E(speed)$	$E(\ell_2)$	$E(lateral)$	$E(longi)$	$E(final\ disp.)$
Keep Straight	CNN-FC	**88.80**	-0.26	83.61	0.78	1.33	0.22	1.27	2.60
	CNN-LSTM	89.08	**-0.13**	82.62	0.88	1.35	0.25	1.28	2.53
	CNN-LSTM+State	91.17	-0.33	**85.68**	**0.30**	**0.77**	**0.20**	**0.70**	**1.54**
Turn Left	CNN-FC	91.38	-0.55	68.59	0.88	1.31	0.52	1.11	2.72
	CNN-LSTM	96.40	**-0.33**	70.42	0.91	1.35	0.48	1.16	2.73
	CNN-LSTM+State	**91.01**	-0.50	**74.50**	**0.35**	**0.61**	**0.35**	**0.41**	**1.47**
Turn Right	CNN-FC	90.20	-0.41	67.97	0.74	1.22	0.50	1.00	2.53
	CNN-LSTM	90.49	**-0.23**	68.93	0.70	1.16	0.49	0.94	2.33
	CNN-LSTM+State	**89.10**	-0.70	**73.24**	**0.32**	**0.63**	**0.38**	**0.41**	**1.48**



(a) Lateral position error after 1.0 second on the test sequence.

(b) Longitudinal position error after 1.0 second on the test sequence.

Fig. 7. For every time step, the error for planned lateral and longitudinal position after 1.0 second over the whole sequence are shown in this figure. The predictions are done by the CNN-FC, CNN-LSTM and our full CNN-LSTM+State method. The x-axis shows the time in the sequence in *seconds*. The figure is best viewed in color.

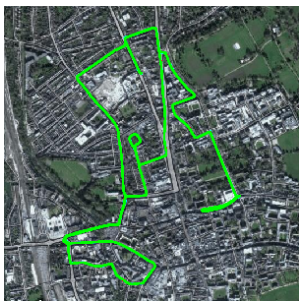


Fig. 8. The green line represents the path for the vehicle in the test sequence, in which there are many turns at intersections suitable to measure the performance of our model.[1]

Additionally, we use the dimensionless jerk (DLJ) metric to calculate the smoothness of the trajectory [27], where $v(t)$ is the movement speed, t_1 and t_2 are the start and end times of the movement. The higher the value of DLJ, the smoother the trajectory is.

$$DLJ \triangleq -\frac{(t_2-t_1)^3}{v_{\text{peak}}^2}\int_{t_1}^{t_2}\left|\frac{d^2v}{dt^2}\right|^2 dt, \quad v_{\text{peak}} \triangleq \max_{t\in[t_1,t_2]} v(t) \tag{5}$$

2) Results: Table I shows the quantitative results from experiments among various models. From the third column of Table. I, it can be seen that the proposed method, CNN-LSTM+State, can run at about 11 Hz, faster than the other methods when turning left and right, which is capable of real-time applications. When the decoder of CNN-FC changed from FC to LSTM, the network CNN-LSTM shows smaller or comparable final displacement error for all scenes and larger IoU for turning left and right. Furthermore, it plans smoother trajectories with higher DLJ. This may be due to the fact that LSTM has an advantage over FC to process temporal information. Finally, by composing image series and past state into our full CNN-LSTM+State network, the performance improves a lot with higher IoU and lower error results on the prediction of speed and positions in the

[1]https://robotcar-dataset.robots.ox.ac.uk/datasets/2015-08-17-13-30-19/

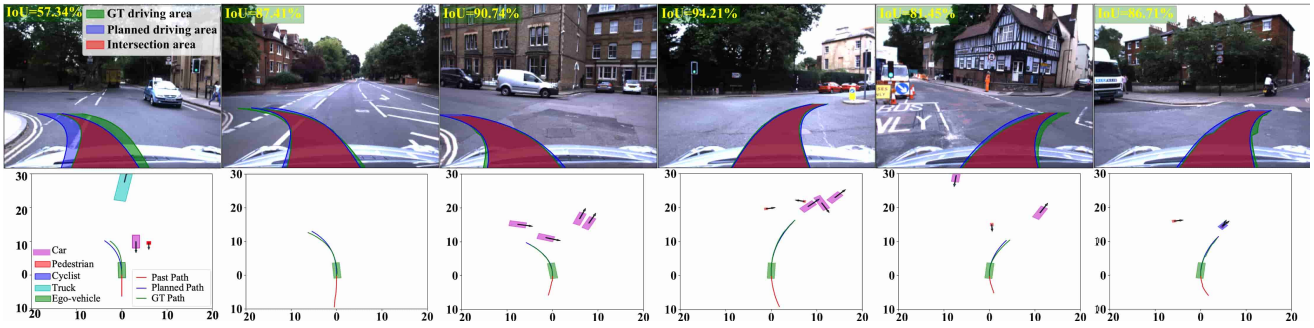


Fig. 9. Results for turning left and right at different intersections by our CNN-LSTM+State model. Row 1 shows the driving areas generated by trajectories. Row 2 shows the top-down view. The horizontal axis indicates the lateral coordinates and the vertical axis indicates the longitudinal coordinates, both in *meters*. The figure is best viewed in color.

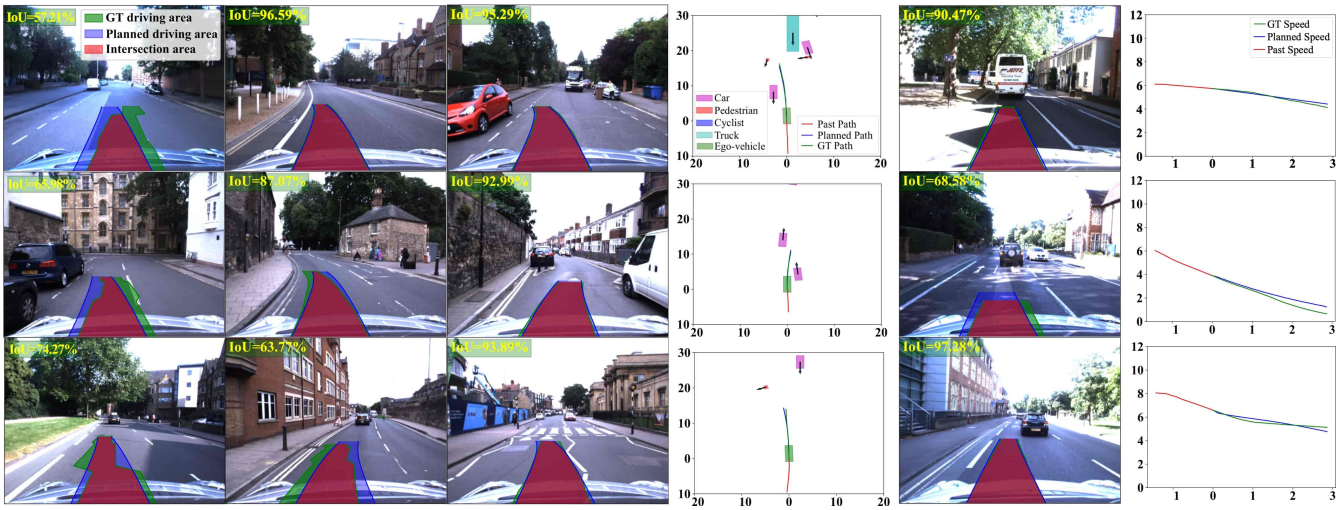


Fig. 10. Some special cases for the qualitative evaluation. Our CNN-LSTM+State model generates smoother trajectories than ground truth when the GPS+INS data deviates slightly from the real driving trajectory, which is shown in column (a). Column (b) shows the situation when driving on curved roads. Column (c1) shows the situation when changing lanes for collision avoidance. Column (c2) shows the corresponding top-down view, of which the horizontal axis indicates the lateral coordinates and the vertical axis indicates the longitudinal coordinates, both in *meters*. Column (d1) shows the situation when slowing down for a slow car. Column (d2) shows the corresponding speed information, of which the horizontal axis indicates the time in *seconds* and the vertical axis indicates the speed in *m/s*. The figure is best viewed in color.

three test scenes. This shows the importance of past state information to plan future trajectories.

Fig. 6 shows the distribution of displacement error on three test scenes over the preview time by different models. It is visible that for each scene, the displacement error increases as the preview time increases for the three models, among which the CNN-FC and CNN-LSTM have similar results and our full CNN-LSTM+State method performs best, with lower error medium and more centralized error distribution, especially at the moment of 1.0s in the *turn left* situation.

We also tested our model on a whole consecutive sequence in our dataset for about 20 minutes. The error results for different models are shown in Fig. 7 and the driving path is visualized in Fig. 8. It can be seen intuitively that the CNN-LSTM+State model plans more accurate positions than the other two methods, especially for the prediction on longitudinal positions.

B. Qualitative Evaluation

We further provide some qualitative analysis to investigate the effectiveness of our model. As introduced earlier, we use the positions recorded by GPS+INS data to form the ground truth trajectories. However, they occasionally deviate a little from the real driving path due to instability of GPS signals. In these scenes, our method can plan much smoother trajectories than ground truth, as shown in column *a* of Fig. 10.

When *turning left* or *turning right* at an intersection, our model can plan a trajectory very close to a human driver, as shown in Fig. 9. For the *keep straight* situation, we mainly concentrate on the following three specific tasks:

(A) lane-keeping on curved roads, shown in column *b* of Fig. 10. Note that the curvature of the road surface in this scene is smaller than that when turning left or right.
(B) changing lanes for collision avoidance, shown in column *c1-c2* of Fig. 10.

(C) slowing down for a slow car, shown in column *d1-d2* of Fig. 10.

In situation (A), the planning performance is still reliable when the road becomes winding. In situation (B), the model plans a trajectory to bypass the obstacle ahead and keep going straight. In situation (C), when the vehicle drives near a slow or parked car, the model plans a low speed to ensure safety.

V. Conclusion

To imitate the driving behavior of human drivers as well as avoid the deficiencies of end-to-end control methods, we proposed an end-to-end planning method achieved by a CNN-LSTM network architecture. We used the front-view image stream and state information in the past 1.5 seconds as input, to plan a possible collision-free trajectory containing speed and lateral/longitudinal positions 3.0 seconds in the future. A CNN module was used to extract visual features for comprehending the environment, and an LSTM module was utilized to consider temporal dependencies. The results suggest that our model plans trajectories close to ground truth when turning at various intersections or keeping straight. We also tested our model in three sub-tasks when keeping straight, where the generated future trajectories could indicate the vehicle to bypass the stopped vehicle ahead for collision avoidance, driving on curved roads and slow down for a parked car.

Some limitations still exists in this work such as we ignored the treatment to traffic lights and we did not consider the planning performance under different weather or lighting conditions. To further improve the performance of our model and apply it to practical applications, more labelled driving data could be utilized for training. Some other sensors, such as a thermal imaging camera, could also be added to enlarge the perceptual range [28] and help the network achieve better planning results. Besides, the restored 3-D information for the surrounding road agents could also be valuable for the planning network, which leaves to future work.

References

[1] Y. Sun, M. Liu, and M. Q.-H. Meng, "Improving rgb-d slam in dynamic environments: A motion removal approach," *Robotics and Autonomous Systems*, vol. 89, pp. 110–122, 2017.

[2] T. Sun, Y. Sun, M. Liu, and D.-Y. Yeung, "Movable-object-aware visual slam via weakly supervised semantic segmentation," *arXiv preprint arXiv:1906.03629*, 2019.

[3] Y. Sun, M. Liu, and M. Q.-H. Meng, "Motion removal for reliable rgb-d slam in dynamic environments," *Robotics and Autonomous Systems*, vol. 108, pp. 115–128, 2018.

[4] C. Wang, W. Chi, Y. Sun, and M. Q.-H. Meng, "Autonomous Robotic Exploration by Incremental Road Map Construction," *IEEE Transactions on Automation Science and Engineering*, pp. 1–12, 2019.

[5] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[6] E. Rehder, J. Quehl, and C. Stiller, "Driving like a human: Imitation learning for path planning using convolutional neural networks," in *International Conference on Robotics and Automation Workshops*, 2017.

[7] P. Wu, Y. Cao, Y. He, and D. Li, "Vision-based robot path planning with deep learning," in *International Conference on Computer Vision Systems*. Springer, 2017, pp. 101–111.

[8] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 187–210, 2018.

[9] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[10] J. Jhung, I. Bae, J. Moon, T. Kim, J. Kim, and S. Kim, "End-to-end steering controller with cnn-based closed-loop feedback for autonomous vehicles," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 617–622.

[11] H. M. Eraqi, M. N. Moustafa, and J. Honer, "End-to-end deep learning for steering autonomous vehicles considering temporal dependencies," *arXiv preprint arXiv:1710.03804*, 2017.

[12] F. Codevilla, M. Miiller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–9.

[13] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2174–2182.

[14] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, "1 year, 1000 km: The oxford robotcar dataset," *The International Journal of Robotics Research*, vol. 36, no. 1, pp. 3–15, 2017.

[15] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, 1989, pp. 305–313.

[16] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian, "Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation," *arXiv preprint arXiv:1710.05627*, 2017.

[17] M. Bergqvist and O. Rödholm, "Deep Path Planning Using Images and Object Data," Master's thesis, Chalmers University of Technology, Gothenburg, Sweden, 2018.

[18] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[20] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[21] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow," https://github.com/matterport/Mask_RCNN, 2017.

[22] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7074–7082.

[23] cersar, "3d_detection," https://github.com/cersar/3D_detection, 2019.

[24] J.-R. Chang and Y.-S. Chen, "Pyramid stereo matching network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5410–5418.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[26] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool, "You'll never walk alone: Modeling social behavior for multi-target tracking," in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 261–268.

[27] N. Hogan and D. Sternad, "Sensitivity of smoothness measures to movement duration, amplitude, and arrests," *Journal of motor behavior*, vol. 41, no. 6, pp. 529–534, 2009.

[28] Y. Sun, W. Zuo, and M. Liu, "Rtfnet: Rgb-thermal fusion network for semantic segmentation of urban scenes," *IEEE Robotics and Automation Letters*, 2019.